最小k-部分木問題に対するタブー探索法に基づく近似解法(最適化数理の手法と実際)

AUTHOR(S):

片桐, 英樹; 坂和, 正敏; 西崎, 一郎

CITATION:

片桐, 英樹 ...[et al]. 最小k-部分木問題に対するタブー探索法に基づく近似解法(最適化数理の手法と実際). 数理解析研究所講究録 2005, 1461: 111-125

ISSUE DATE:

2005-12

URL:

http://hdl.handle.net/2433/47955

数理解析研究所講究録 1461 巻 2005 年 111-125

最小k-部分木問題に対するタブー探索法に基づく近似解法

広島大学大学院・工学研究科　片桐 英樹 (Hideki KATAGIRI)
坂和 正敏 (Masatoshi SAKAWA)
西崎 一郎 (Ichiro Nishizaki)
Graduate School of Engineering
Hiroshima University

1　はじめに

最小k-部分木問題とは，ノード集合と重み付アーク集合から構成されるグラフにおいて，重みの総和が最小になるk本のアークから構成される木を求める問題である．この問題はHamacherら[1]によって最初に定式化され，遠隔通信[2]や施設配置[3, 4]，露天採鉱[5]，行列分解[6, 7]などに応用が見られるように現実社会で幅広く見受けられる問題である．

最小k-部分木問題はNP-困難な組合せ最適化問題であることが証明[8, 9]されており，アークの重みが{1,2,3}の3種類の場合やグラフが完全グラフおよび平面グラフの場合でもなおNP-困難であることが示されている[9]．これまでに分枝限定法[10]や分枝カット法[11]に基づく厳密解法も提案されているが，大規模な問題に対しては一般に実用時間内に厳密解を求めることは困難であるため，効率の良い近似解法の構築が重要な課題となっている．

Blumら[12]は複数のメタヒューリスティック手法に基づく近似解法を提案し，ベンチマーク問題[13]による数値実験を通して，アーク密度が高いグラフでかつkが大きい場合には，タブー探索法[15]に基づく解法が他手法に対して優れていることを示した．

本研究では，Blumらの手法の問題点を指摘し，問題点を解決するためにBlumらとは別の近傍構造と探索法を提案する．さらに，ベンチマーク問題に対する数値実験を通して，Blumらの手法と提案手法を精度および計算時間の両面で比較を行い，提案手法の有効性を示す．

2　最小k-部分木問題の定式化

ノード集合Vとアーク集合Eからなるグラフ$G=(V,E)$において，k-部分木 T_kが

$$T_k \in G, \quad k \leq |V|-1$$

と定義されるとき，最小k-部分木問題は次のように定式化される．

$$\begin{aligned} \text{minimize} \quad & f(T_k)=\sum_{e\in E(T_k)} w(e) \\ \text{subject to} \quad & T_k \in \mathcal{T}_k \end{aligned}$$

ただし，fは実数値関数，$\mathcal{T}_k$はGに含まれる全てのT_kの集合，$E(T_k)$はT_kを構成するアーク集合，$w(e)$はアークeの重みである．この問題は，$(k-1)$個のノードを繋いで木を構成するときに，最小の重み和をもつアーク集合を求めるという組合せ最適化問題であり，問題の規模が小さい場合には数え上げによって容易に最適解を求めることが可能である．しかし，ノード数やアーク数が増加するについて数え上げで最適解を求めることは困難になり，分枝限定法による解法[10]や分枝カット法に基づく解法[11]を用いたとし

ても規模が大きくなると実用時間内に最適解を求めることは不可能になる．組合せ最適化問題に対しては，従来より遺伝的アルゴリズムを初めとするメタヒューリスティック手法の研究が行われ，最小 k-部分木問題に対しても Blum ら [12] によって，進化的計算手法，アントコロニー最適化法，タブー探索法に基づいた近似解法を提案し，複数のベンチマーク問題 [13] に適用することにより，ノード数に対してアーク数が多い密なグラフや k が大きい場合に対してタブー探索法の優位性を示されている．Blum らの手法においては，Jornsten ら [14] によるアーク集合に基づく近傍構造を用いており，アークの追加および削除に対してそれぞれ逆戻りの削除および追加をタブーとして課している．

本研究では，Blum らのタブー探索法を用いた解法の問題点を指摘し，ノードの追加と削除に対してタブーを課しながら部分グラフに対して最小木問題を解くという局所的探索を組み込んだタブー探索法に基づく解法を提案する．また，ベンチマーク問題に対して，数値実験を行い，解の精度の向上と計算時間の短縮を行う．

次節では，まず一般的なタブー探索法の概要について述べる．

3 タブー探索法の概要

近傍 $N(\boldsymbol{x})$ の全体あるいは一部の中で，現在の解 $\boldsymbol{x}$ 以外で目的関数値が最も良い解を次の解として選択する場合，現在の解 $\boldsymbol{x}$ が局所最適解ならば，$\boldsymbol{x}$ から他の解 $\boldsymbol{x}' \in N(\boldsymbol{x})$ に移った後に同様の操作を行うと再び元の $\boldsymbol{x}$ に戻る．タブー探索法ではこのような逆戻りあるいはいくつかの解を経由して戻る巡回を避けるため，タブーリスト（短期メモリ）と呼ばれる解の遷移に関する情報の集合 T を用意し，T に含まれる遷移を禁止（タブー）して，$N(\boldsymbol{x}) \setminus (\{\boldsymbol{x}\} \notin T)$ 内の最良の解へ移動するものとする．

手順 1 初期解 $\boldsymbol{x}$ を生成し，タブーリスト T を初期化する．

手順 2 $N(\boldsymbol{x}) \setminus (\{\boldsymbol{x}\} \notin T)$ において，最良解 $\boldsymbol{x}'$ を見つけ，$\boldsymbol{x} := \boldsymbol{x}'$ とする．

手順 3 終了条件がみたされれば暫定解を出力して探索を終了する．そうでなければ，タブーリスト T を更新した後ステップ 2 に戻る．

手順 2 において，タブーリスト T には解そのものではなく最近の近傍操作において移動の前後で値の変わった変数などを記憶し，値の変更そのものや変更前の値への逆戻りを禁止する．このような禁止規則を保持し続けると，移動できる解がいずれ無くなるため，タブー期間と呼ばれるパラメータ t_{tabu} を用意し，タブーリストに入って t_{tabu} 回反復したらリストから取り除く．

手順 3 における終了条件としては，1）あらかじめ定められた反復回数で終了する，2）あらかじめ定められた反復回数の間に暫定解の更新がなければ終了する，などが用いられる．

また，タブー探索法ではタブーリストを使用するだけでなく，特定の決定変数を変更した頻度や決定変数がある値をとり続けた期間の長さなどの探索履歴を記憶し，良質な解が含まれると思われる領域の集中的な探索や未探索領域への遷移を行うことが有効であるとされている．このような探索履歴の記憶を中長期メモリと呼ばれ，代表的なものとして，ある変数が解の移動において変更あるいは特定の値をとっていた頻度を保存する「頻度メモリ」がある．頻度メモリの利用法の一つとして，ある特定の変数の値が過去の探索で頻繁に変更されている場合は，長い周期での巡回が起こっていると判断し，その変数の値を変更することに対してペナルティを与えるものがある．これは探索解の多様化を目的としており，通常，解の評価値にそのような変数のペナルティを重み付きで加えることで実現されるが，ペナルティが大きすぎると良い解を探索する能力がかえって低くなる．したがって，ペナルティの重みを小さくするか，探索の多様化が必要であると考えられるとき（局所最適解からの脱出を行うときや暫定解が比較的長い間更新されないときなど）のみにペナルティを与える等の方法がとられる．

4 最小 k-部分木問題に対する従来解法

Blum らが最小 k-部分木問題に対して提案したタブー探索法に基づく近似解法の概要は次のようになる.

手順 1 (初期解の生成) ランダムに選択したノードを出発点として木を成長させ, k-部分木になった時点で初期解とする.

手順 2 (局所的探索) 近傍において, 現在の k-部分木の葉ノードから出ているアーク集合の中で重みが最大のアークを一本削除して $(k-1)$-部分木を生成した後, アーク集合の中で重みが最小のアークを 1 本追加することでより良い解への遷移を行う. 遷移する解が存在しなければ全てのタブーを解除し, 現在の解を初期解として手順 2 を繰り返す.

(終了条件) 得られた解が過去に探索された最良解を更新しなければ, タブー期間をある規則で増加させ, 手順 2 へ戻る. 最良解を更新すれば, 初期のタブー期間に戻す. タブー期間がある閾値を越えるならば, 手順 1 へ戻る. ある一定期間, 最良解が更新されなければ探索を終了する.

以下に Blum らの手法について詳細を述べる.

4.1 近傍構造とタブーリスト

k-部分木 T_k から任意のアーク e を一本削除することでできる全ての $(k-1)$-部分木 T_{k-1} からなる集合を $E_{NH1}(T_k)$ とする. また, E_{NH2} を次のように定義する.

$$E_{NH2}(T_{k-1}) \leftarrow \{e = \{v, v'\} \in E(G) \mid v \in V(T_{k-1}) \ \mathbf{XOR}\ v' \in V(T_{k-1})\}$$

ここで, $E(G)$ はグラフ G に含まれているアーク集合, $V(T_{k-1})$ は T_{k-1} に含まれているノード集合を示す.

T_{k-1} に対して, $E_{NH2}(T_{k-1})/\{e\}$ に含まれるアークを一本追加してできるすべての k-部分木集合を k-部分木 T_k における近傍 $N(T_k)$ とする.

また, タブーリストとして $InList$ と $OutList$ を用意し, それぞれ遷移において T_k から削除されたアークと削除後できた T_{k-1} に追加されたアークの番号を一定期間保存する.

4.2 アルゴリズムの詳細

タブー探索法を用いた Blum らによる従来手法のアルゴリズムは次のようになる.

Algorithm TS(C. Blum, M. J. Blesa)
InitializeParameters($tt_{min}, tt_{max}, tt_{inc}, tl_{ten}, nic, nic_{max}$)
InitializeTabuLists ($Inlist, OutList, tl_{ten}$)
T_k^{cur} := GenerateInitialSolution()
$T_k^{gb} := T_k^{cur}, T_k^{rb} := T_k^{cur}$
while termination condtions not met **do**
T_k^{new} :=FirstInprovingNeighbor($N(T_k^{cur}), Inlist, OutList$)
if$T_k^{new} \neq$ NULL
UpdateTabuLists($T_k^{cur}, T_k^{new}, InList, OutList$)
$T_k^{cur} := T_k^{new}$
Update($T_k^{cur}, T_k^{rb}, T_k^{gb}, nic$)
if $nic > nic_{max}$ **then**
bf if $tl_{ten} + tt_{inc} > tt_{max}$ **then**

```
            PerformRestart()
        else
            tl_ten := tl_ten + tt_inc
        end if
    end if
else
    PerformRestart()
end if
end while
output: T_k^gb
```

次に各関数の説明をする．

(1)InitializeParameters$(tt_{min}, tt_{max}, tt_{inc}, tl_{ten}, nic, nic_{max})$**:**

各変数に初期値を決定する．ノード集合 V，アーク集合 E を持つグラフ G，アークの重み w，k からなる問題 $(G=(V,E),w,k)$ が与えられたとき，

$$tt_{min} := \min\left\{\left\lfloor\frac{|V|}{5}\right\rfloor, |V|-k, k\right\}$$

$$tt_{max} := \left\lfloor\frac{|V|}{3}\right\rfloor$$

$$tt_{inc} := \left\lfloor\frac{tt_{max}-tt_{min}}{4}\right\rfloor + 1$$

$$tt_{min} := \max\{tt_{inc}, 200\}$$

と設定する．また，現在のタブー期間を表す tl_{ten} を

$$tl_{ten} := tt_{min}$$

とする．

(2)InitializeTabuLists $(Inlist, OutList, tl_{ten})$**:**

2つのタブーリスト $InList$，$OutList$ を空の状態，すなわち，タブーである属性が無い状態へと戻す．

(3)GenerateInitialSolution() :

初期解の新たに k-部分木を生成する．まず，ランダムにアーク $e=v,v' \in E$ を選び，1本のアーク e と2つのノード v，v' からなる 1-部分木 T_1 を作る．その後，次の操作を $k-1$ 解繰り返すことで k-部分木 T_k を作る．さらに，$e := argmin\{w(e') \mid e' \in E_{NH}(T_t)\}$ となるアーク e を T_t に加えることで T_{t+1} を作る．

(4)FirstInprovingNeighbor$(N_{leaf}, Inlist, OutList)$**:**

現在の k-部分木 T_k^{cur} の隣接点が選択される．$T_k \in N_{leaf}(T_k^{cur})$，すなわち，$T_k = T_k^{cur} - e_{out} + e_{in}$ である全ての k-部分木集合の中で次の2つの条件のどちらかを満たした k-部分木が選択可能となる．

(a) $e_{in} \notin InList$ かつ $e_{out} \notin OutList$

(b) $f(T_k) < f(T_k^{gb})$，ここで，T_k^{gb} は探索中に見つかった最良解である．

ここで，条件 (2) は特別選択基準である．

現在の探索解の近傍内で，現在の解よりも良い目的関数値を与える解が見つかれば，すぐに次の探索点として移動する．もし，近傍内に s よりも良い目的関数値をもつ解が存在しないならば，近傍内を

全て探索してその中で最良目的関数値を与える解を次の探索解とし，近傍内に遷移可能な解が存在しなかった場合は，戻り値としてNULLを返す．

(5)UpdateTabuLists($T_k^{cur}, T_k^{new}, InList, OutList$)**:**

隣接点 $T_k^{new} = T_k^{cur} - e_{out} + e_{in}$ が選択された後，タブーリストである $InList$ と $OutList$ の更新が行われ，e_{in} が $OutList$ へ，e_{out} が $InList$ へそれぞれ追加される．両タブーリストは期間 tl_{ten} の first-in-first-out リストであり，ごく最近に現在の木から削除された（追加された）アークを，一定期間の間に再び追加される（削除される）ことを禁止する．

(6)Update($T_k^{cur}, T_k^{rb}, T_k^{gb}, nic$)**:**

$f(T_k^{cur}) < f(T_k^{rb})$ ならば，$T_k^{rb} := T_k^{cur}$ とする．逆に，$f(T_k^{cur}) < f(T_k^{rb})$ ならば nic の値を1増加する，ここで，nic は (Re)Start 後に最良解が連続して更新されなかった回数を表す．

(7)PerformRestart():

近傍内に遷移可能な解が存在しなかった場合，あるいはタブー期間 tl_{ten} がその最大値 tt_{max} に達した場合，Restart として次のアルゴリズムが実行される．

$tl_{ten} := tt_{min}$

InitializeTabuLists($InList, OutList, Tl_{ten}$)

T_k^{cur} :=GenerateInitialSolution()

$T_k^{rb} := T_k^{cur}$

$nic := 0$

4.3 Blum らの手法の問題点と完全案

Blum らの近傍探索では，現在の解である k-部分木において，葉ノードから出ているアークを1本削除した後に，新たにアークを1本追加している．この場合，アーク数の少ない疎なグラフに対しては，現在の k-部分木に含まれるノード集合以外のノードから出ているアークを追加することによって次々と探索が進む可能性が高い．しかし，アーク数が多い密なグラフに対しては，ノード集合同士をつなぐアーク数が多いために同一のノード集合内での探索が繰り返し行われて，新たなノード集合を含む解への遷移速度が遅くなり，結果的に十分な探索が進まない可能性もある．

したがって，本研究では，アークではなくノードに着目した追加と削除を行い，固定されたノード集合から生成されるグラフに対して Prim 法などの最小木問題を解く手法を適用して新たな k-部分木を求め，次には未選択のノード集合を含む探索点への遷移が進むように改良を行う．また，Blum らの手法では組み込まれていなかった適応メモリ戦略を組み込み，探索解の多様化を図ることで，安定して良質の解が得られる手法の開発を試みる．

5 提案アルゴリズム

本節では，最小 k-部分木に対するタブー探索法を用いた提案手法について説明する．ここでは，従来手法と同様に2つのタブーリストを用意し，$InList$ では遷移の際に削除したノード，$OutList$ では追加したノードを保存する．

手順1 (初期解の生成) ランダムに選択したノードを出発点として木を成長させ，k-部分木になった時点で初期解とする．

手順 2 (局所的探索) 現在の解に含まれる葉ノード集合において，最大の重みのアークをもち，かつ削除禁止でない葉ノードを削除することで $(k-1)$ 木を生成する．次に，生成された $(k-1)$ 木に繋がるアークをもつノード集合の中で，最小の重みのアークをもち，かつ追加禁止でないノードを追加することで新たな k-部分木を生成する．

手順 3 (局所的最適解の導出) 新しく生成された解に含まれるノード集合とそのノード集合同士を繋ぐアークから生成されるグラフに対してプリム法を適用し，手順 2 へ戻る．ある一定の間，最良解が更新されなければ手順 4 に進む．

手順 4 (多様化戦略) 過去の探索解に含まれた回数が多いノードを削除し，回数が少ないノードを追加することを一定期間行い，手順 1 へ戻る．

(終了条件) ある一定期間，最良解が更新されなければ探索を終了する．

提案手法のアルゴリズムを次に示す．

Algorithm TS(Katagiri)
InitializeParameter($tl_{in}, tl_{out}, Freq[\,]$)
InitializeTabuList($InList, OutList$)
T_k^{cur} :=GenerateInitialSolution()
$T_k^{gb} := T_k^{cur}$, $AspirationCriteria := f(T_k^{cur})$
UpdateFrequency($T_k^{cur}, Freq[\,]$)
while termination condition not met **do**
  $nic_{int} := 0$
  while $nic_{int} < 50$ **do**
    if termination condition not met **then**
      T_k^{new} :=LocalSearch($N(T_k^{cur}), InList, OutList$)
      if $T_k^{new} \neq$NULL then
        $T_k^{cur} := T_k^{new}$
        UpdateFrequency($T_k^{cur}, Freq[\,]$)
        if $AspirationCriteria > f(T_k^{new})$ **then**
          nic_{int}:=0
          Update($T_k^{cur}, T_k^{gb}, AspirationCriteria$)
        else
          $nic_{int} := nic_{int} + 1$
        end if
      else
        PerformRestart()
      end if
    else
      $nic_{int} := 50$
    end if
  end while
  $nic_{diver} := 0$
  while $nic_{diver} < k$ **do**
    if termination condition not met **then**
      T_k^{new} :=Diversification($N(T_k^{cur}), InList, OutList, Freq[\,]$)

if $T_k^{new} \neq$NULL **then**
$nic_{diver} := nic_{diver} + 1$
$T_k^{cur} := T_k^{new}$
UpdateFrequency($T_k^{cur}, Freq[\,]$)
Update($T_k^{cur}, T_k^{gb}, AspirationCriteria$)
else
$nic_{diver} := k$
PerformRestart()
end if
else
$nic_{diver} := k$
end if
end while
end while
output: T_k^{gb}

次にアルゴリズム TS(Katagiri) の中で使用されている関数について説明する.

(1)InitializeParameter($tl_{in}, tl_{out}, Freq[\,]$):

変数 tl_{in}, tl_{out} はそれぞれ $InList$, $OutList$ におけるタブー期間, 配列 $Freq[\,]$ はノードの使用頻度を表す. $Freq[v] := n$ は, ノード v が今までの探索において n 回 k-部分木を構成するノードとして使用されたことを示す. また, すべてのノード $v \in V$ に対して $Freq[v] := 0$ とする.

(2)InitializeTabuList($InList, OutList$):

2 つのタブーリスト $InList$, $OutList$ を空の状態, すなわち, タブーである属性が無い状態へと戻す.

(3)UpdateFrequency($T_k^{cur}, Freq[\,]$):

すべての $v \in V(T_k^{cur})$ に対して, $Freq[v] := Freq[v] + 1$ とする.

(4)LocalSearch($N(T_k^{cur}, InList, OutList)$):

現在の木 T_k^{cur} から, 削除することにより T_{k-1} となるアーク集合 E_{NH1} 内で, タブーでなく, かつ最も重みの大きいアークを削除アーク e_{out} とし, $E_{NH2}(T_{k-1})/\{e_{out}\}$ のなかで, タブーでなく, かつ最も重みの小さなアーク, あるいは特別選択基準を満たし, かつ最も重みの小さなアークを追加アーク e_{in} とする. また, 追加アーク e_{in} の端のノードのうち T_{k-1} に含まれていないノードを v^{new} とする ($v^{new} \notin V(T_{k-1})$). このとき, ノード v^{new} と T_{k-1} を結ぶアークの数が 1 本のみならば, $T_k^{new} = T_{k-1} + e_{in}$ とする (図 1 参照). しかし, 2 本以上の場合, T_{k-1} を構成するノード集合 $V(T_{k-1})$ と追加ノード $\{v^{new}\}$ の和集合 $\{v^{new}\} \cup V(T_{k-1})$ にプリム法を適用することで, これらのノード集合を用いた場合の最適な k-部分木を生成し, T_k^{new} とする (図 2 参照). ただし, 削除アーク e_{out}, あるいは追加アーク e_{in} が無かった場合は, NULL を返す.

(5)LocalSearch ($N(T_k^{cur}), InList, OutList$):

局所的探索を行う関数 LocalSearch のアルゴリズムは次のように構成する.

Algorithm LocalSearch($N(T_k^{cur}), InList, OutList$)

$T_k^{new} := NULL$

$E_{out} := E_{NH1}(T_k^{cur})$, $e_{out} := NULL$

while $E_{out} \neq \phi$ **then**

choose $e \in E_{out}$

UpdateOutArk$(e_{out}, e, OutList)$

$E_{out} := E_{out}/\{e\}$

end while

if $e_{out} \neq NULL$ **then**

$T_{k-1}^{new} := T_k^{cur} - e_{out}$

$E_{in} := E_{NH}(T_{k-1}^{new})/\{e_{out}\}$, $e_{in} := NULL$

while $E_{in} \neq \phi$ **do**

choose $e \in E_{in}$

UpdateInArk$(e_{in}, e, T_{k-1}^{new}, InList)$

$E_{in} := E_{in}/e$

end while

if $e_{in} \neq NULL$ **then**

$E_{new} := \{e = \{v^{new}, v'\} \in E(G) \mid v' \in V(T_{k-1}^{new})\}$

if $|E_{new}| \neq 1$

T_k^{new} :=PrimMethod$(V(T_{k-1}), v^{new}, G)$

else

$T_k^{new} := T_{k-1}^{new} + e_{in}$

end if

end if

end if

output:T_k^{new}

ただし，アルゴリズム Local Search の中で使用されている関数は次のように設定する．

UpdateOutArk$(e_{out}, e, OutList)$**:**

$e_{out} = NULL$ のとき，$e \notin OutList$ ならば，$e_{out} := e$ とする．$e_{out} \neq NULL$ のとき，$e \notin OutList$ かつ $w(e) > w(e_{out})$ ならば $e_out := e$ とする．

UpdateInArk$(e_{in}, e, T_{k-1}^{new}, InList)$**:**

$e \in InList$ ならば，特別選択基準を満たしている $(AspirationCriteria > f(T_{k-1}^{new}) + w(e))$ かどうかを調べる．満たしているならタブーでないとする．

$e_{in} = NULL$ のとき，e がタブーでないならば $e_{in} = e$ とする．$e_{in} \neq NULL$ のとき，e がタブーでなく，かつ $w(e_{in}) > w(e)$ ならば $e_{in} := e$ とする．

(6)PrimMethod$(V(T_{k-1}), v^{new}, G)$**:**

ノード集合 $V(T_{k-1}) \cup \{v^{new}\}$ に対してプリム法を適用し，新たな木 $T_k^{new} \in G$ を生成する．

(7)Update($T_k^{cur}, T_k^{gb}, AspirationCriteria$):

もし，$AspirationCriteria > f(T_k^{cur})$ ならば，$AspirationCriteria = f(T_k^{cur})$ とする．同様に T_k^{gb} に対しても，$f(T_k^{gb}) > f(T_k^{cur})$ ならば，$f(T_k^{gb}) = f(T_k^{cur})$ とする．

(8)PerformRestart():

探索中に，近傍内に遷移可能な解が存在しなかった場合，Restart として次のアルゴリズムが行われる．

InitializeTabuLists($InList, OutList$)

T_k^{cur} :=GenerateInitialSolution()

$AspirationCriteria = T_k^{cur}$

$nic := 0$

(9)Diversification($T_k^{cur}, InList, OutList, Freq[\ ]$):

ここでは，頻度メモリを用いて探索解の多様化を行っている．現在の解 T_k^{cur} から，削除することで T_{k-1} となるアークの集合を E_{out} としたとき，$e_{out} = argmax\{Freq(e) \mid e \in E_{out}\}$ を削除アークとする．次に，$e_{in} = argmin\{Freq(e) \mid e \in E_{NH}(T_{k-1})/e\}$ を追加アークとし，新たな k-部分木 $T_k^{new} = T_{k-1} + e_{in}$ を生成する．

5.1 提案手法の特徴

提案手法の最も大きな特徴は，固定されたノード集合に対して，そのノード集合とそれらを繋ぐアーク集合のみから構成される部分グラフに対して最小木問題を解く点にある．最小 k-部分木問題は重みの総和が最小になる k-部分木を求める問題であるが，仮に最小 k-部分木を構成する $(k-1)$ 個のノード集合を特定することができれば，その部分グラフに対して最小木問題を解くアルゴリズムを適用することで最適解を求めることができる．したがって，本提案手法では，固定された部分グラフに対して最小木問題を解いた後，一つのノードだけ削除および追加を行い，異なる部分グラフに対して再び最小木問題を解くアルゴリズムを適用することを繰り返し行っている．この手法は繰り返し最小木問題を解くために計算時間を要するようにも思えるが，後で述べる数値実験の結果が示すように寧ろ従来手法よりも計算時間が短い．これは最小木問題が多項式時間で解けることも大きな要因の一つであると考えられる．

ゆえに，最小 k-部分木問題に限らず，属性を固定する部分グラフに対する子問題が多項式時間で解けるような組合せ最適化問題に対しては，属性の削除および追加に対してタブーを課すというアイデアに基づくアルゴリズムも有力であると考えられる．

6 数値実験

従来手法と提案手法を比較するために，Blum らのベンチマーク問題（問題 1 から 7）および新たに作成したベンチマーク問題（問題 8 および 9）に対して，提案手法と従来のタブー探索法をそれぞれ 30 回ずつ適用し，精度および計算時間について比較を行った．また，実験環境として，CPU: Celeron 2.4GHz, C-Compiler: Microsoft Visual C++ 6.0 を使用した．表において，TS(Blum) および EC(Blum) はそれぞれ Blum らによるタブー探索法および進化的計算手法に基づく従来解法であり，TS(Katagiri) が本研究で提案したタブー探索法に基づく近似解法である．

Blum らのベンチマーク問題においては，密なグラフが欠けている（彼らが密なグラフとして挙げているものも疎なグラフになっている）ため，ここでは新たにベンチマーク問題として問題 8 および 9 を生成し，数値実験を行っている．

実験結果より，本提案手法は従来法と比較して，疎なグラフに対しては同程度，密なグラフに対しては従来法よりも良い精度の解が得られていることがわかる．また，提案手法は問題8よりも問題9に対して特に優位性が際立つため，グラフの密度が大きいほど有効であると考えられる．Blumらは密度が高く，kが大きい場合に対してタブー探索法が他手法に比べて有効であることを示しているが，9つのベンチマーク問題の結果を見る限りでは本提案手法はBlumらの解法に替わる有力な手法であると予想される．計算時間についても問題7以外では全て短縮されているが，問題7では従来法と比較して1.5倍以上の計算時間を要しており，問題の構造によっては従来法よりも悪くなる場合もあることを示唆するものと考えられる．

7 おわりに

本研究では，最小k-部分木問題に対して，タブー探索法を用いた新たな解法を提案し，いくつかの数値実験により解の精度と計算時間を比較し，提案手法が従来法よりも有効であることを示した．しかし，今回行ったベンチマーク問題は数が限られており，提案手法が従来手法よりも有効であると結論付けることはできない．Blumらはベンチマーク問題として260個（グラフとしては20個で，それぞれに対して13種類のkの値を設定）用意し[13]，タブー探索法，アントコロニー最適化法，進化的計算手法などの複数の手法を比較し，9ヶ月間に及ぶ数値実験を行ってそれぞれの精度と最適値を導出しており，その結果はWeb上に掲載されている．今回の数値実験では，Blumらのベンチマーク問題の中に含まれている格子グラフおよび正規グラフに対しては行っておらず，今後はBlumらの用意した全てのベンチマーク問題および新たなベンチマーク問題に対して精度および計算時間の面で比較を行うと共に実験結果を分析し，さらなる改善を行う予定である．

また，本研究では詳細を述べなかったが，タブー期間の値がパフォーマンスに大きく影響することもわかっている．適切なタブー期間はノード数やアーク数だけでなく，アーク密度やkの値にも依存すると考えられるため，今後は適切なタブー期間の設定法に関しても研究を進めたいと考えている．

参考文献

[1] H.W. Hamacher, K. Jorsten, F. Maffioli, Weighted k-cardinality trees, Technical Report 91.023, Politecnico di Milano, Dipartimento di Elettronica, Italy, 1991.

[2] N. Garg, D. Hochbaum, An $O(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane, Algorithmica, Vol.18, No.1, 111-121, 1997.

[3] L.R. Foulds, H.W. Hamacher, J. Wilson, Integer programming approches to facilities layout models with forbidden areas, Annals of Operations Research, Vol. 81, pp. 405–417, 1998.

[4] L.R. Foulds, H.W. Hamacher, A new integer programming approach to restriced facilities layout problems allowing flexible facility shapes, Technical Report 1992-3, University of Waikato, Department of Management Science, 1992.

[5] A.B. Philpott, N.C. Wormald, On the optimal extraction of ore from an open-cast mine, New Zealand: University of Auckland, 1997.

[6] R. Borndorfer, C. Ferreira, A. Martin, Matrix decomposition by branch-and-cut, Technical Report, Konrad-Zuse-Zentrum fur Informationstechnik, Berlin, 1997.

[7] R. Borndorfer, C. Ferreira, A. Martin, Decomposing matrices into blocks, SIAM Journal on Optimization, Vol.9, No.1, 236–269, 1998.

[8] M. Fischetti, H.W. Hamacher, K. Jornsten, F. Maffioli, Weighted k-cardinality trees: complexity and polyhedral structure, Networks, Vol. 24, 11–21, 1994.

[9] M.V. Marathe, R. Ravi, S.S. Ravi, D.J. Rosenkrantz, R. Sundaram, Spanning trees short or small, SIAM Journal on Discrete Mathematics, Vol. 9, No.2, 178–200, 1996.

[10] S.Y. Cheung, A. Kumar, Efficient quorumcast routing algorithms, Proceedings of INFOCOM'94, Los Alamitos, USA, Silver Spring, MD: IEEE Society Press, 1994.

[11] J. Freitag, Minimal k-cardinality trees, Master's thesis, Department of Mathematics, University of Kaiserslautern, Germany, 1993.

[12] C. Blum, M.J. Blesa, New metaheuristic approaches for the edge-weighted k-cardinality tree problem, Computers & Operations Research, Vol. 32, pp. 1355–1377, 2005.

[13] KCTLIB. http://iridia.ulb.ac.be/ cblum/kctlib/, 2003.

[14] K. Jornsten, A. Lokkentangen, Tabu search for weighted k-cardinality trees, Asia-Pacific Journal of Operational Research, Vol. 14, No.2, 9–26, 1997.

[15] F. Glover, M. Laguna, Tabu Search, Dordrecht: Kluwer Academic Publishers, 1997.

表 1: 問題: g25-4-01.dat[13] ノード:25 アーク:50 k:20 TimeLimit:5

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	219.0	219.0	219.0
平均目的関数値	219.0	219.0	219.0
最悪目的関数値	219.0	219.0	219.0
平均計算時間 (秒)	0.0943	0.9456	0.003
最良目的関数値を得た回数	30	30	30

表 2: 問題: g50-4-01.dat[13] ノード:50 アーク:98 k:20 TimeLimit:10

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	460.0	460.0	460.0
平均目的関数値	460.0	462.3	460.0
最悪目的関数値	462.0	464.0	460.0
平均計算時間 (秒)	1.8760	3.5293	0.1683
最良目的関数値を得た回数	30	3	30

表 3: 問題: g75-4-05.dat[13] ノード:75 アーク:150 k:20 TimeLimit:10

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	284.0	284.0	284.0
平均目的関数値	284.0	284.0	284.0
最悪目的関数値	284.0	284.0	284.0
平均計算時間 (秒)	0.0750	1.0710	0.0060
最良目的関数値を得た回数	30	30	30

表 4: 問題: g100-4-01.dat[13] ノード:100 アーク:200 k:20 TimeLimit:10

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	363.0	363.0	363.0
平均目的関数値	363.0	363.0	363.0
最悪目的関数値	363.0	363.0	363.0
平均計算時間 (秒)	0.4605	1.8945	0.4054
最良目的関数値を得た回数	30	30	30

表 5: 問題: g200-4-01.dat[13] ノード:200 アーク:400 k:20 TimeLimit:10

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	308.0	308.0	308.0
平均目的関数値	308.0	308.0	308.0
最悪目的関数値	308.0	308.0	308.0
平均計算時間 (秒)	0.3631	3.0691	0.2313
最良目的関数値を得た回数	30	30	30

表 6: 問題: g400-4-01.dat[13] ノード:400 アーク:800 k:20 TimeLimit:20

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	253.0	253.0	253.0
平均目的関数値	253.0	253.0	253.0
最悪目的関数値	253.0	253.0	253.0
平均計算時間 (秒)	0.23441	1.5383	0.0229
最良目的関数値を得た回数	30	30	30

表 7: 問題: g1000-4-01.dat[13] ノード:1000 アーク:2000 k:20 TimeLimit:25

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	263.0	263.0	263.0
平均目的関数値	264.4	267.0	264.0
最悪目的関数値	267.0	270.0	267.0
平均計算時間 (秒)	7.2517	10.6303	12.5521
最良目的関数値を得た回数	19	10	22

表 8: ノード:200 アーク:2000 k:100 TimeLimit:10

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	105.0	185.0	105.0
平均目的関数値	107.1	232.3	105.9
最悪目的関数値	113.0	260.0	107.0
平均計算時間 (秒)	3.1865	11.6389	1.3085
最良目的関数値を得た回数	15	1	10

表 9: ノード:100 アーク:4950 k:80 TimeLimit:10

	TS(Blum)	EC(Blum)	TS(Katagiri)
最良目的関数値	183.0	1214.0	183.0
平均目的関数値	207.0	1392.1	183.0
最悪目的関数値	338.0	1574.0	183.0
平均計算時間 (秒)	4.0341	34.6302	0.6642
最良目的関数値を得た回数	19	1	30

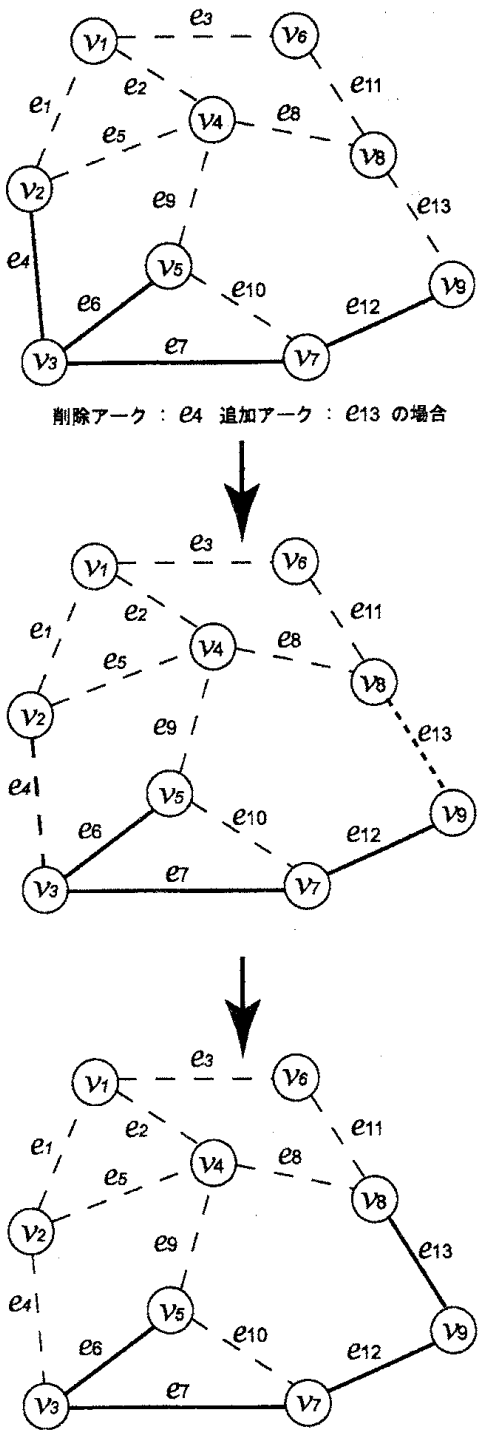

図 1: 局所的探索で Prim 法を行わない場合

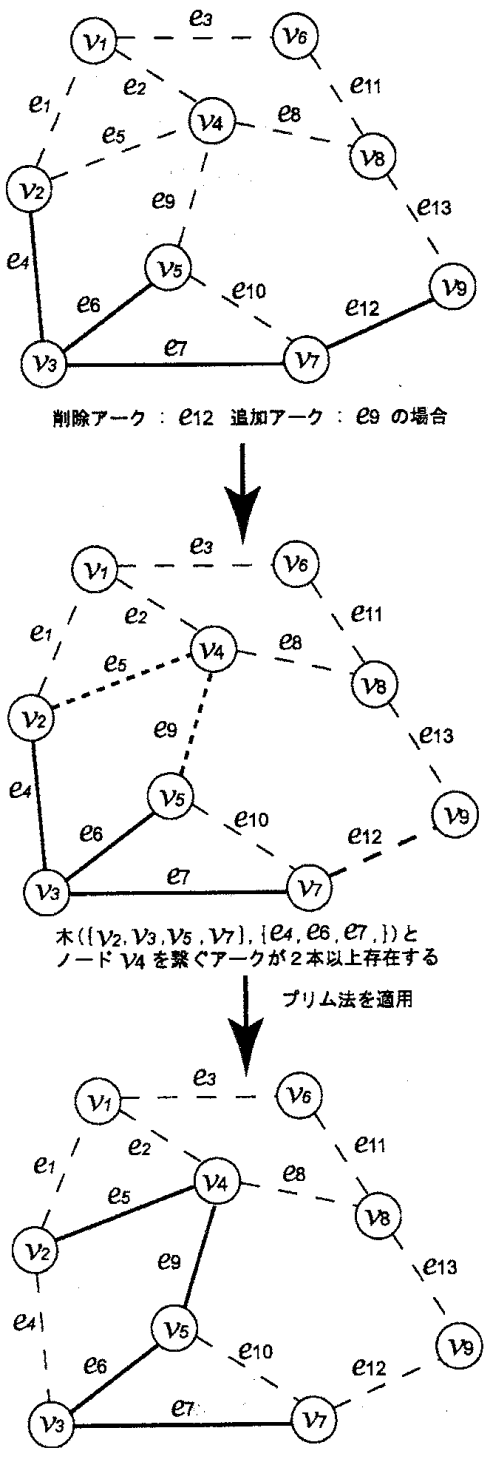

図 2: 局所的探索で Prim 法を行う場合